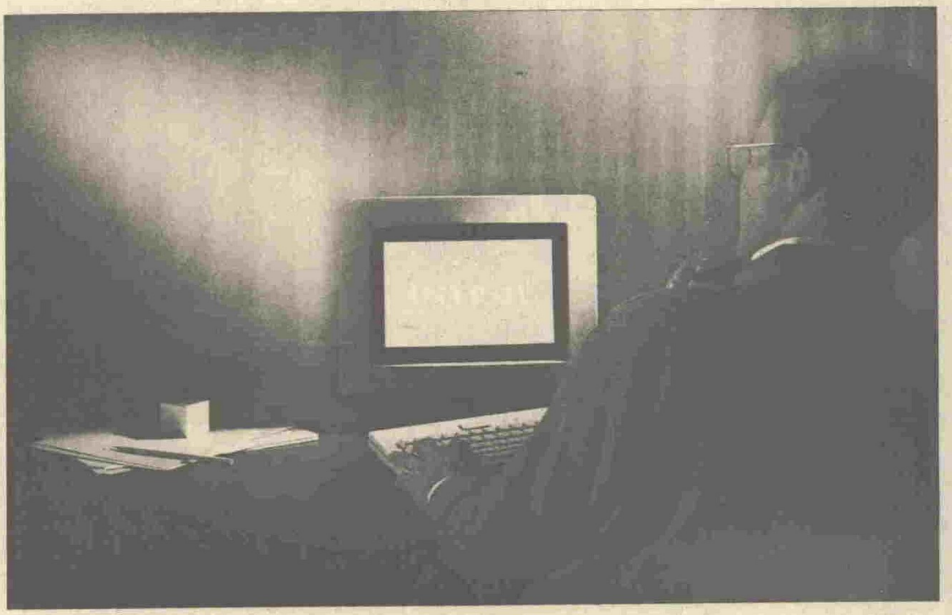
อนันต์ กิตติบรรจุ

Declaration
เคล็ดวิชาในภาษาซี

ให้ความกระจ่างในเรื่องการประกาศในภาษาซี เพื่อให้ผู้รักภาษาซีเขียนโปรแกรมได้อย่างมีประสิทธิภาพ



ภาษาซีเป็นภาษาที่ได้รับความนิยมอย่างสูงในปัจจุบัน อันเนื่องมาจากเป็นภาษาที่ได้รวมเอาจุดเด่นจากภาษาขั้นสูงมารวมกับภาษาขั้นต่ำ และมีโครงสร้างภาษาที่ดีสะดวกสบายในการเขียนโปรแกรม กล่าวคือ มีไลบรารีฟังก์ชันที่มีประโยชน์มากมาย เช่น ฟังก์ชันที่จัดการด้านอินพุต/เอาต์พุต สตริง เมมโมรีและอื่น ๆ ทำงานได้รวดเร็วและจัดได้ว่าโค้ดมีขนาดเล็ก และการที่เป็นภาษาที่ค่อนข้างจะไม่ขึ้นกับระบบเครื่อง (portable) คือสามารถนำซอร์สโค้ดไปคอมพายล์บนเครื่องระดับต่างกันได้โดยแก้ไขเพียงเล็กน้อย แต่รูปแบบของภาษา (syntax) บางส่วนซับซ้อน โดยเฉพาะเรื่องของพอยน์เตอร์ (pointer) อะเรย์ และการประกาศ (Declaration) ในรูปแบบต่าง ๆ เป็นต้น เนื่องจากขาดความเข้าใจในรูปแบบภาษาซีอย่างแท้จริงนี่เองทำให้โปรแกรมเมอร์ไม่สามารถที่จะเขียนโปรแกรมได้อย่างมีประสิทธิภาพเต็มที่ แม้แต่ผู้ที่ชำนาญแล้วก็ตาม ในจุดประสงค์ของบทความนี้จึงต้องการให้ความกระจ่างกับผู้อ่านที่สนใจและรักภาษาซีในเรื่องของการประกาศและการตีความหมายในการประกาศนั้น ๆ เริ่มต้นขอให้ดูการประกาศข้างล่างนี้

```
struct vtag far * (far const * var[5] ();
```

อย่าเพิ่งตกใจไปก่อนนะครับเมื่อเห็นการประกาศเช่นนี้ เมื่อผู้อ่านค่อย ๆ อ่านจนจบบทความนี้แล้ว จะต้องเข้าใจและสามารถที่ตีความและเขียนการประกาศได้ดีขึ้น

รูปแบบของการประกาศ (Declaration Syntax)

ในการที่จะใช้ภาษาใด ๆ ก็ตาม คุณจะต้องรู้เกี่ยวกับโครงสร้าง (Structure) และรูปแบบ (Syntax) ก่อน ในการประกาศของภาษาซีจะมีการจัดโครงสร้างและรูปแบบของการประกาศเป็นดังรูปที่ 1

จะเห็นว่าการประกาศหนึ่ง ๆ จะมีตัวระบุบอก (Specifier) อยู่หลายตัว แต่จริง ๆ แล้วสามารถใช้เพียงแค่ type และตัวประกาศ (declarator) ก็ได้ แต่ก็ยังมีกรณีที่ตัวแปรถูกระบุอยู่นอกฟังก์ชัน สามารถมีเพียงตัวแปรได้ เช่น ในการประกาศ "extern x" แม้จะถือว่าตัวแปร x เป็นข้อมูลชนิด integer ก็ตาม แต่มันไม่ใช่วิธีการเขียนภาษาซีที่ถูกต้อง เขาเรียกกันว่า "bad programming style" ซึ่งอาจทำให้เกิดปัญหาข้อผิดพลาดหรือ bug ได้ง่าย ๆ

ทฤษฎีของการประกาศ

โปรแกรมเมอร์ส่วนใหญ่สามารถที่จะตีความการประกาศนี้ได้ถูกต้อง คือ

```
int i ;
char * ptr:
```

แต่ในการประกาศตัวแปรที่ซับซ้อน ส่วนใหญ่ยังต้องใช้วิธีเดาหรือลองผิดลองถูก แต่หากใช้ตามทฤษฎีแล้วมันไม่ใช่เรื่องยากเลย เพียงแต่ให้เข้าใจลำดับการทำงาน (precedence) ของแต่ละโอเปอเรเตอร์เหมือนกับที่คุณผู้อ่านใช้กับการให้ค่าหรือหาค่านิพจน์คณิตศาสตร์ (expression) นั่นเองสำหรับลำดับจะเป็นดังนี้คือ

( ) หรือ [ ] มีลำดับสูงที่สุดและจากซ้ายไปขวา
* ลำดับต่ำสุด

ในหนังสือ Brian W. Kernighan and Dennis M. Ritchie, The C Programming Language (Prentice-Hall, Inc., 1978) หรือ ที่เรามักเห็นในตำราหลาย ๆ เล่มเขียนสั้น ๆ ว่า K & R ได้ให้กฎสำหรับการอ่านและการเขียนการประกาศไว้ ดังมีรายละเอียดตามรูปที่ 2

จากรูปที่ 2 เป็นกฎสำหรับการอ่านและเขียนของการประกาศตัวแปรของ K & R โดยก่อนที่จะวิเคราะห์ต้องใส่วงเล็บตามลำดับการทำงานเสียก่อน และสำหรับตัวอย่างการนำกฎนี้ไปใช้ขอให้ดูรูปที่ 3 ซึ่งได้ใช้วงเล็บเพื่อเรียงลำดับการทำงานไว้ และการประกาศโดยวงเล็บก็สามารถใช้ได้ (แต่ไม่นิยมใช้กัน) ส่วนในรูปที่ 4 เป็นตัวอย่างการขยายความตามกฎของลำดับการทำงาน และการใช้กฎในรูปที่ 2 แต่จะไม่วิเคราะห์โดยใช้วงเล็บ ใช้วิธีหาตำแหน่งของตัวแปร แล้วจึงมองเครื่องหมายทางขวามือ คือ ( ) หรือ [ ] ประกอบกันในการตีความ โดยรายละเอียดการวิเคราะห์ก็ศึกษากับกรณีตัวอย่างในรูปที่ 4 และขอให้พึงระลึกข้อสังเกตเพิ่มเติมจากกฎดังนี้คือ

1. ใช้อะเรย์ของฟังก์ชันไม่ได้ แต่สามารถใช้อะเรย์ของพอยน์เตอร์ไปยังฟังก์ชันได้ เช่น ใช้ "int x[5]( )" ไม่ได้ แต่ใช้ int (*x[5]) ( ) ได้

รูปแบบมาตรฐานการประกาศของภาษา C

มีรูปแบบคือ
"Storage-class type qualifier declarator = initializer"
โดย Storage-class เป็น 1 ในเหล่านี้คือ
typedef, extern, static, auto และ register
Type เป็นชนิดของตัวแปร ได้แก่
void, char, short, int, long, float, double, signed, unsigned, struct.., union... และ typedef type (เป็น type จากการประกาศ typedef)
qualifier หรือ บางตำราใช้ modifier เป็นตัวบอกคุณสมบัติของตัวแปร ได้แก่ const และ volatile
declarator หรือตัวประกาศ ซึ่งเป็นส่วนที่ประกอบด้วย identifier (อาจเป็นชื่อตัวแปรหรือ ชื่อฟังก์ชัน เป็นต้น) และ "*" หรือ "( )" ซึ่งส่วนนี้จะมีหรือไม่มีก็ได้
initializer เป็นค่าที่ให้ในตอนเริ่มต้น

หมายเหตุ สามารถที่จะนำวงเล็บมาใช้แยกกลุ่ม ในการเชื่อมกลุ่มต่างกันได้

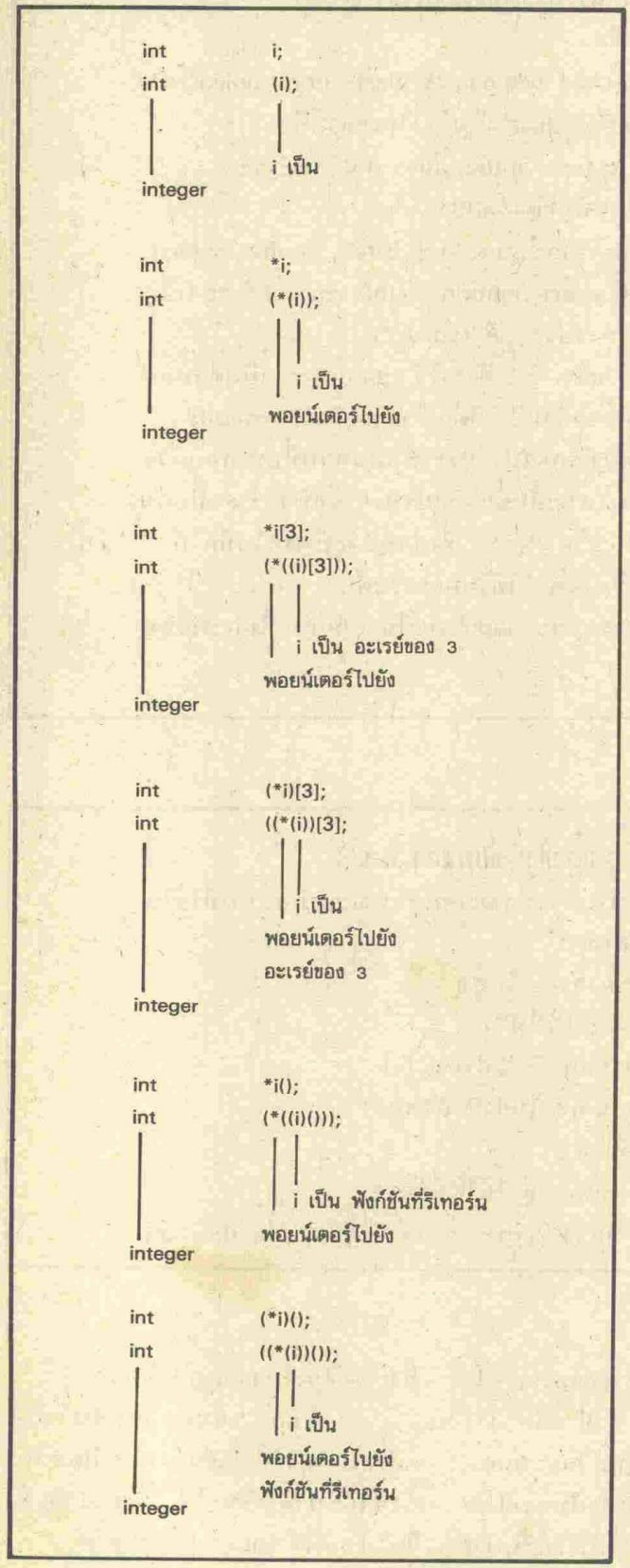
รูปที่ 1

กฎสำหรับการอ่านและการเขียนของ K & R

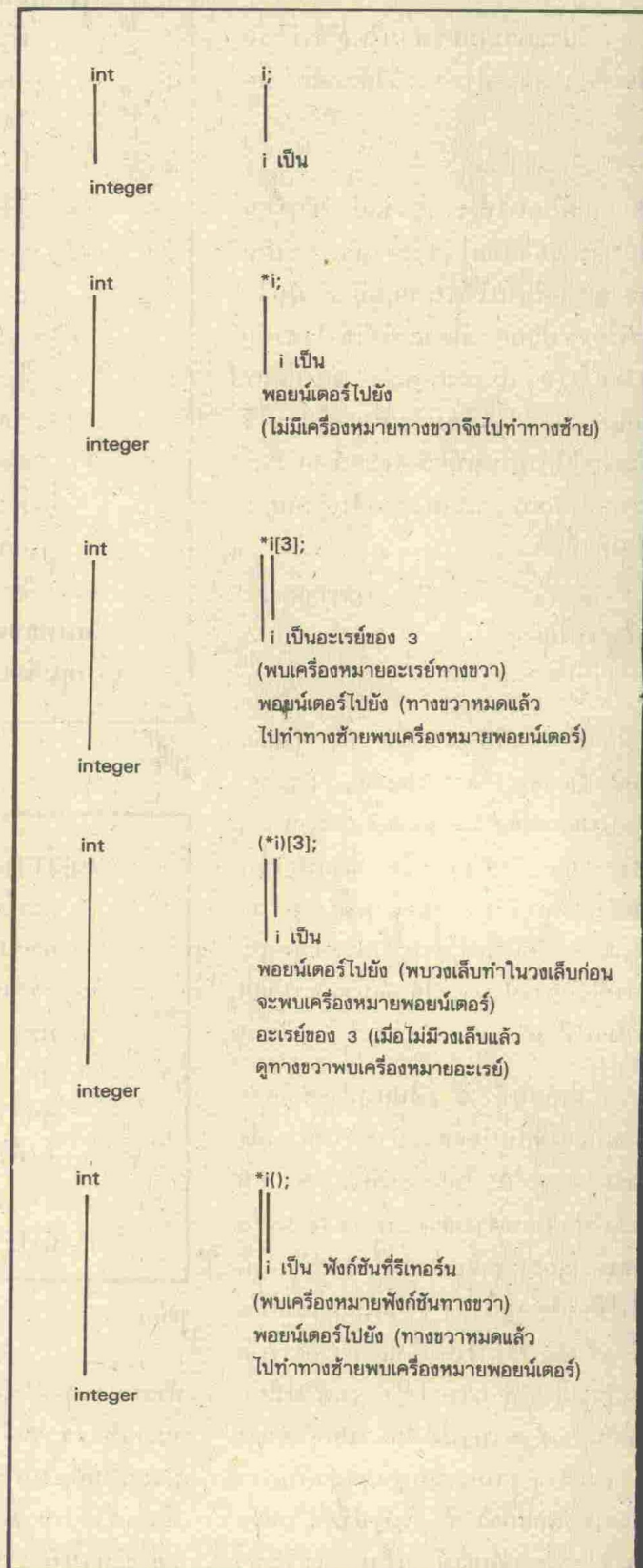
1. การใช้วงเล็บในการประกาศมีลักษณะเหมือนกับที่ใช้ในนิพจน์คณิตศาสตร์
2. หาตำแหน่งของวงเล็บในสุด
3. พูดว่า "ชื่อตัวแปรเป็น"
   พูดว่า "อะเรย์ของ x" ถ้าพบ [x]
   พูดว่า "พอยน์เตอร์ไปยัง" ถ้าพบ *
4. ไปที่วงเล็บถัดไป
5. ถ้ายังมีวงเล็บเหลืออยู่ ให้ไปที่ข้อ 3.
6. ถ้าไม่มีแล้ว พูดว่า "type" จากซ้ายไปขวา เช่น short int

รูปที่ 2

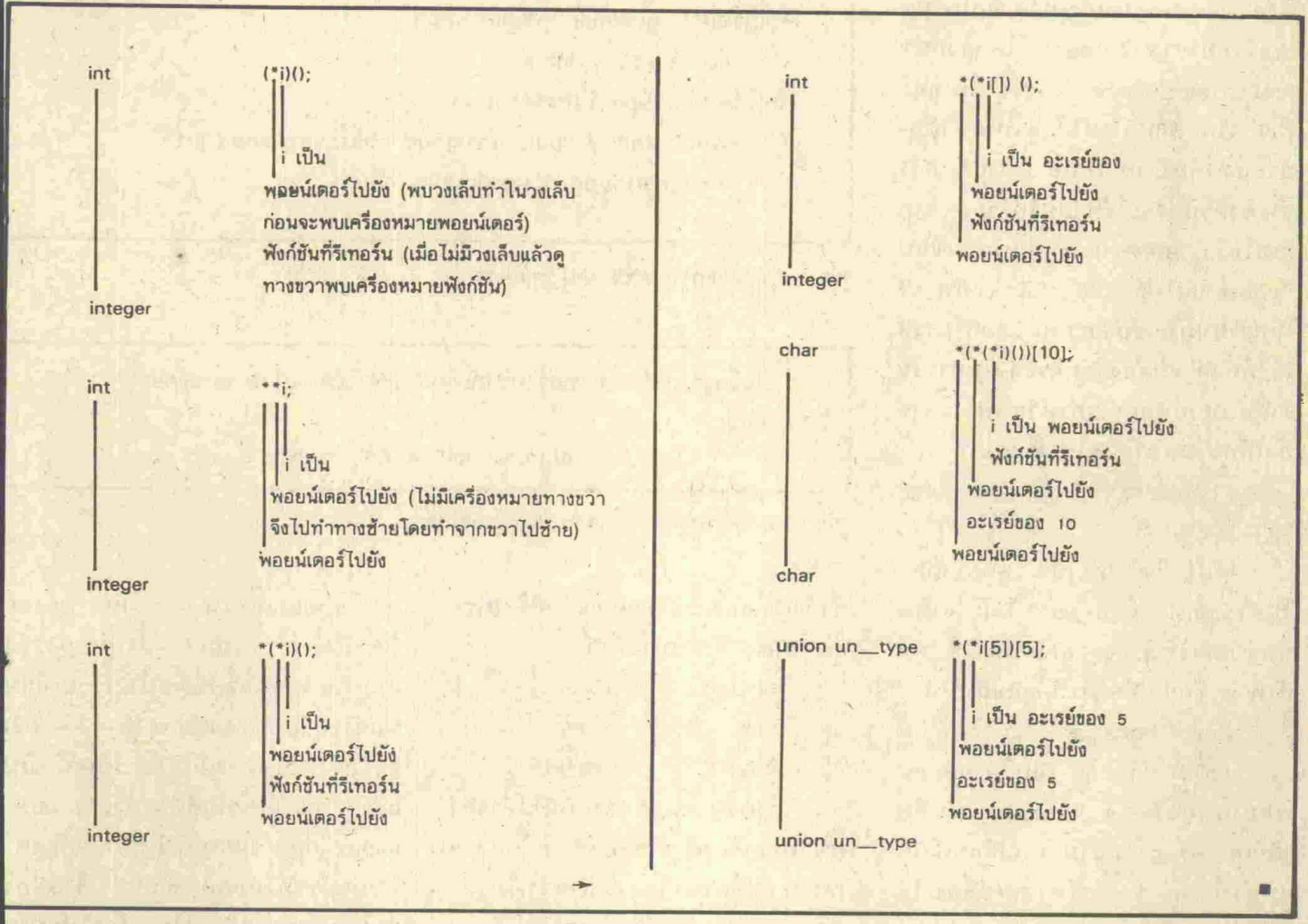
รูปที่ 3 แสดงการตีความการประกาศโดยใช้วงเล็บ



รูปที่ 4 แสดงการตีความการประกาศโดยไม่ใช้วงเล็บ



รูปที่ 4 (ต่อ)



2. ฟังก์ชันไม่สามารถที่จะรีเทอร์นอะเรย์ แต่จะรีเทอร์นพอยน์เตอร์ไปยังอะเรย์ได้ เช่น ใช้ "int x( )[ ]" ไม่ได้ แต่ใช้ "int *x( )" ได้

3. ฟังก์ชันไม่สามารถที่จะรีเทอร์นฟังก์ชันอื่นได้ เช่น ใช้ "int x( )( )" ไม่ได้

การเพิ่มมาตรฐาน ANSI

คณะกรรมการ X3J11 ซึ่งมีหน้าที่บัญญัติมาตรฐานสำหรับภาษาซี โดยเฉพาะได้เพิ่มเติมคีย์เวิร์ดจากมาตรฐาน K & R และ ANSI ดังรูปที่ 5

สำหรับคีย์เวิร์ดส่วน qualifier จากรูปที่ 5 const จะเป็นตัวบอกว่าคุณสมบัติของ identifier ว่าไม่สามารถทำการเพิ่ม ลด หรือให้ค่าได้ โดยมันสามารถให้ค่าได้เพียงครั้งเดียวจากตอนประกาศตัวแปรในครั้งแรก (ค่า initializer) เช่น

```
const int x = 7;
char *const str = "Micro";
```

ส่วน volatile จะใช้งานตรงข้ามกับ const กล่าวคือ เป็นตัวบอกว่าตัวแปรนั้น ๆ สามารถที่จะเปลี่ยนแปลงค่าได้ แต่ไม่ใช่ภายในส่วนโปรแกรม แต่เป็นส่วนภายนอก เช่น ถูกเปลี่ยนในอินเตอร์รัพต์รูทีนหรือในการทำอินพุต/เอาต์พุตพอร์ต และเป็นการบังคับไม่ให้คอมพายเลอร์ทำการ optimize code ด้วย ดังเช่น

```
volatile int real__time__clock;
real__time__clock = 0;
while (real__time__clock <
TIME__WAIT);
```

จากตัวอย่างข้างบนสมมติว่าตัวแปร real__time__clock เป็นตัวแปรที่ถูกรูทีนไทม์เมอร์อินเตอร์รัพต์เปลี่ยน-

แปลงตลอดเวลา ถ้าเราไม่กำหนดให้ตัวแปรนี้เป็น volatile แล้ว คอมพายเลอร์จะทำการ optimize โดยตัวแปร real_time_clock อาจใช้รีจิสเตอร์อ้างอิงใน while ลูปได้ ทำให้ผิดวัตถุประสงค์ทันที แต่ถ้าเป็น volatile การอ้างอิงตัวแปรจะใช้ตัวแปรอ้างอิง (จากเมมโมรี) โดยตรงเลย ส่วนการเขียน "const int i" หรือ "int const i" ให้ความหมายเช่นเดียวกัน และสามารถใช้ได้ทั้งคู่ แต่ต้องระมัดระวังในการใช้งานกับพวกพอยน์เตอร์ด้วย เพราะอาจทำให้ความหมายผิดไป ดังเช่น

```
"const int *p" หรือ "int const *p"
```

หมายถึง "p เป็น พอยน์เตอร์ไปยัง constant integer" โดย p เป็นพอยน์เตอร์ที่สามารถเปลี่ยนค่าได้ แต่ค่าที่ p ชี้อยู่เปลี่ยนไม่ได้ แต่ถ้าเป็น

```
"int * const p"
```

จะหมายถึง p เป็น constant พอยน์เตอร์ไปยัง integer นั่นคือ พอยน์เตอร์ p ไม่สามารถเปลี่ยนค่าได้ แต่ค่าที่ p ชี้อยู่สามารถเปลี่ยนได้ ตัวอย่างการใช้ constant พอยน์เตอร์ที่เราจะเห็นกันบ่อย ๆ ก็มี

```
int printf (char * const format,...);
```

เพื่อจะบอกว่า ฟังก์ชัน printf รับสตริง format ไปใช้เท่านั้นแต่จะไม่ทำการเปลี่ยนค่าของสตริง format

การเพิ่มมาตรฐานโดยไมโครซอฟต์ซีและเทอร์โบซี

จากรูปที่ 6 ซึ่งเป็นมาตรฐานที่ 2 ค่ายยักษ์ใหญ่แห่งคอมพายเลอร์ C ได้บัญญัติขึ้นมาเพื่อขยายการใช้งานออกไป

คีย์เวิร์ดส่วน qualifier ที่เพิ่มเติมได้แก่
const และ volatile
คีย์เวิร์ดส่วน type ที่เพิ่มเติมได้แก่
void, signed char, unsigned char, unsigned int, unsigned long, long double, enum...

รูปที่ 5 การเพิ่มมาตรฐาน ANSI (ANSI Extensions)

ไมโครซอฟต์ซีและเทอร์โบซีได้เพิ่มเติมคีย์เวิร์ดในส่วน qualifier ดังนี้
near, far, huge, cdecl, pascal, interrupt

รูปที่ 6 การเพิ่มมาตรฐานโดยไมโครซอฟต์ซีและเทอร์โบซี

ซึ่งคีย์เวิร์ดเฉพาะตัวบอกคุณสมบัติ (type qualifier) ที่เพิ่มมาได้แก่

far	cdecl
near	pascal
huge	interrupt

สำหรับ 3 ตัวข้างนั้นใช้อ้างอิงเรื่องพอยน์เตอร์ ทั้งนี้ก็เพื่อให้เกิดความเหมาะสมในการใช้งานสำหรับไมโครโปรเซสเซอร์ของอินเทล ตระกูล 8086/88, 80286/88,.. ซึ่งมีเรื่องของเซกเมนต์และออฟเซต เพื่อใช้ระบุในการอ้างโค้ดหรือดาต้า ตามขนาดของเมมโมรีโมเดลที่ใช้ โดยที่

near pointer คือการอ้างพอยน์เตอร์ จะอ้างเพียงแค่ค่าออฟเซตหรือขนาดของพอยน์เตอร์ จะเป็น 16 บิต เท่านั้น ส่วน far และ huge pointer การอ้างอิงจะอ้างทั้งเซกเมนต์และออฟเซต หรือขนาดพอยน์เตอร์จะเป็น 32 บิต แต่การอ้างอิงจะแตกต่างกันเล็กน้อย เช่น ถ้า p1 ถูกชี้ไปที่ตำแหน่ง 0000:0120, p2 ชี้ไปยัง 0010:0020 และ p3 ชี้ไปยัง 0012:0000 จะเห็นว่า ค่าเอปโซลูตแอดเดรส จะเป็น 00120 (20 บิต) เท่ากันหมดแต่ถ้า p1, p2, p3 เป็น far พอยน์เตอร์เมื่อมาเปรียบเทียบกันค่าจะไม่เท่ากันเช่น if (p1 = = p2) จะให้ค่าความจริงเป็นเท็จ แต่ถ้าเป็น huge พอยน์เตอร์แล้วจะทำการ normalize ก่อน โดยจะทำให้ค่าพอยน์เตอร์เป็นเอปโซลูตแอดเดรสแล้ว จึงแปลงเซกเมนต์และออฟเซตใหม่ โดยให้ขนาดของออฟเซตมีค่าระหว่าง 0-F (ฐาน 16) เท่านั้น เช่น

0040:0056 จะถูกแปลงเป็น 00456 แล้วจึงแปลงเป็น 0045:0006 เป็นต้น นั่นคือ ถ้าเปรียบเทียบ p1, p2 และ p3 คราวนี้จะได้เท่ากันหมดจริง ๆ

คราวนี้มาดูอีก 2 ตัวคือ cdecl กับ pascal ซึ่งจะใช้นำหน้าฟังก์ชัน เพื่อให้มันสามารถถูกเรียกใช้จากภาษาอื่น ๆ ได้ โดยให้คอมพายเลอร์ทำการเยนเนอเรตโค้ด ให้เหมือนกับเป็นภาษานั้น ๆ เช่นถ้าใช้ "int pascal func (long, int)" (อ่านว่า func เป็นปาสคาลฟังก์ชันที่รีเทอร์น integer) จะหมายถึง

ฟังก์ชัน func จะถูกเปลี่ยนชื่อเป็น FUNC และจะไม่มีการเยนเนอเรตคาแรกเตอร์ "__" (อ่านว่า underscore) นำหน้า และการส่งผ่านพารามิเตอร์จะเป็นแบบภาษาปาสคาล ส่วน cdecl จะใช้เพื่อบังคับให้คอมพายเลอร์ทำการเยนเนอเรตโค้ดโดยมี underscore นำหน้า, การส่งผ่านพารามิเตอร์เป็นแบบภาษาซี เป็นต้น คือถ้าคอมพายล์โปรแกรมโดยใช้ออปชันเป็นแบบปาสคาล แต่ฟังก์ชันที่มี cdecl นำหน้าจะยังคงเป็นรูปแบบซีเช่นเดิม ดังเราจะเห็นฟังก์ชันโปรโตไทพ์ของ STDIO.H จะมี cdecl นำหน้าเพื่อให้การเรียกใช้ฟังก์ชันของ STDIO.H จะส่งพารามิเตอร์ให้กันอย่างถูกต้อง สำหรับลำดับการส่งผ่านพารามิเตอร์นั้น ถ้าเป็นปาสคาลจะส่งโดยมีลำดับจากซ้ายไปขวา แต่ถ้าเป็นซีจะทำจากขวาไปซ้าย ดังตัวอย่างเช่น

```
int cdecl min(int v1, int v2);
int pascal min (in v1, int v2);
```

ถ้าพิจารณาจากสแต็กจะเป็นดังรูปที่ 7

ถ้าเป็น cdecl หลังจากรูทีนที่เรียกฟังก์ชันนี้เสร็จแล้วต้องทำการเพิ่ม sp ไปอีก 4 (หรือมีโค้ดภาษาแอสเซมบลีเป็น add sp, 4) เพื่อเป็นการเคลียร์ค่าพารามิเตอร์ที่อยู่ในสแต็ก แต่ถ้าเป็นภาษาปาสคาลแล้วฟังก์ชัน min จะเป็นตัวเคลียร์ค่าพารามิเตอร์ในสแต็กเอง (ใช้ ret 4)

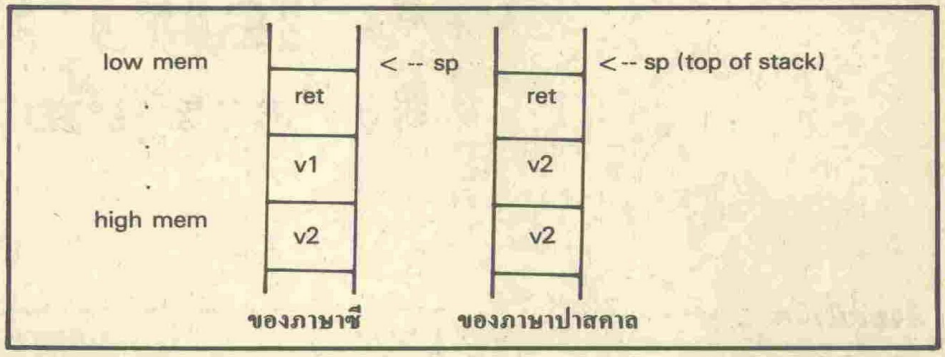
การใช้พอยน์เตอร์ไปยังฟังก์ชันของ cdecl และ pascal เช่น

```
int (pascal * func) (char x);
```

หรือจะใช้

```
int pascal (* func) (char x);
```

ก็ได้ (อ่านว่า func เป็นพอยน์เตอร์ไปยังปาสคาลฟังก์ชันที่รีเทอร์น integer โดยรับค่า char) ถ้าต้องการให้ func ชี้ไปยังฟังก์ชัน test คือ int pascal test (char ch); ได้โดย "func = test" เป็นต้น



รูปที่ 7 แสดงลำดับการส่งพารามิเตอร์ผ่านทางสแต็ก

ส่วนตัวคีย์เวิร์ดสุดท้ายที่จะพูดในบทความนี้ก็คือ interrupt ซึ่งเป็นตัวบอกคุณสมบัติ (qualifier) ที่ใช้สำหรับการเขียนรูทีนอินเตอร์รัพต์

ตัวอย่างการใช้เช่น "void interrupt new_timer ( )" คอมพายเลอร์จะทำการเยนเนอเรตโค้ดพิเศษเพิ่มให้ คือ จะทำการเก็บรีจิสเตอร์ทุกตัว คือ ax, bx, cx, dx, es, ds, si, di และ bp ตามลำดับ โดยการ push ไว้ในสแต็ก (เพื่อเป็นการเก็บสถานะทุกอย่างก่อนที่จะมาในรูทีน) เนื่องจากเป็นรูทีนอินเตอร์รัพต์มันจึงสามารถถูกเรียกได้ทุกครั้งที่เกิดอินเตอร์รัพต์ ดังนั้นในรูทีนจะมีส่วนที่เปลี่ยนดาต้าเซกเมนต์กลับมาของตัวเองเพื่อให้สามารถอ้างอิงตัวแปรได้ถูกต้อง และเมื่อสิ้นสุดรูทีนจะ pop ค่ารีจิสเตอร์จากสแต็กกลับมาและจะกลับไปทำงานดังเดิมโดยคำสั่ง IRET (interrupt return)

คราวนี้ก็มาถึงตอนท้ายแล้วล่ะครับ ลองทดสอบว่าเข้าใจเพียงใดโดยลองตีความการประกาศข้างล่างนี้ดู

```
struct vtag far * (far const * var [5]) (int far * ips);
```

มีความหมายว่า "var เป็นอะเรย์ของ 5 constant far พอยน์เตอร์ไปยังฟังก์ชันที่รีเทอร์น far พอยน์เตอร์ไปยัง struct vtag โดยรับค่า far พอยน์เตอร์ไปยัง integer" และลองทำอีกซักตัวอย่างนะครับ

```
int (far * v (int (far * fp [5]) (int far * ip)))(int far * ip);
```

มีความหมายว่า "v เป็น ฟังก์ชันที่รับค่าอะเรย์ของ 5 far พอยน์เตอร์ไปยังฟังก์ชัน (ของ fp) ที่รีเทอร์น integer โดยรับค่า far พอยน์เตอร์ไปยัง integer และ v จะรีเทอร์น far พอยน์เตอร์ไปยังฟังก์ชันที่รีเทอร์น integer และรับค่า far พอยน์เตอร์ไปยัง integer เช่นกัน"

จากตัวอย่างหลังที่ให้ลองทำดูนั้นจริง ๆ แล้วไม่ควรจะเขียนเช่นนี้หรอกครับ เพราะนอกจากเข้าใจยากแล้วอาจใช้ผิดความหมายก็ได้ ถ้าผู้อ่านเข้าใจในการประกาศนี้ก็แสดงว่าฝีมือเข้าขั้น (เว่อร์) แล้วครับ กว่าที่ผู้เขียนจะเข้าใจก็หืดแทบขึ้นทีเดียว ทางที่ดีควรใช้ typedef ช่วยทำไปทีละขั้นจะดูง่ายกว่าและแน่นอนกว่าด้วย ท้ายนี้ผู้เขียนหวังว่า บทความนี้คงช่วยให้ความเข้าใจและได้อะไรดี ๆ มั่งนะครับในการประกาศและการตีความ แค่นี้ล่ะครับ ถ้ามีโอกาสผู้เขียนจะหาอะไรที่ปวดหัวมาฝากต่อครับ ซาโยนาระ...